International Journal of Information Security manuscript No.
(will be inserted by the editor)

Kostas G. Anagnostakis · Michael B. Greenwald · Sotiris Ioannidis · Angelos D. Keromytis

COVERAGE: Detecting and Reacting to Worm Epidemics Using Cooperation and Validation

Abstract Cooperative defensive systems communicate and cooperate in their *response* to worm attacks, but determine the presence of a worm attack solely on local information. *Distributed* worm detection and immunization systems track suspicious behavior at multiple cooperating nodes to determine *whether* a worm attack is in progress. Earlier work has shown that cooperative systems can respond quickly to day-zero worms, while distributed detection systems allow detectors to be more conservative (*i.e.* paranoid) about potential attacks because they manage false alarms efficiently.

In this paper we present our investigation into the complex tradeoffs in such systems between communication costs, computation overhead, accuracy of the local tests, estimation of viral virulence, and the fraction of the network infected before the attack crests. We evaluate the effectiveness of different system configurations in various simulations. Our experiments show that distributed algorithms are better able to balance effectiveness against worms and viruses with reduced cost in computation and communication when faced with false alarms. Furthermore, cooperative, distributed systems seem more robust against malicious participants in the immunization system than earlier cooperative but non-distributed approaches.

Kostas G. Anagnostakis
Institute for Infocomm Research,
21 Heng Mui Keng Terrace, Singapore 119613
E-mail: kostas@i2r.a-star.edu.sg

Michael B. Greenwald
Bell Labs, Lucent Technologies, Inc.,
600-700 Mountain Ave., Murray Hill, NJ 07974, USA
E-mail: greenwald@research.bell-labs.com

Sotiris Ioannidis
Computer Science Department,
Stevens Institute of Technology,
Hoboken, NJ 07030, USA
E-mail: si@cs.stevens.edu

Angelos D. Keromytis
Department of Computer Science,
Columbia University,
1214 Amsterdam Ave. New York, NY 10027, USA
E-mail: angelos@cs.columbia.edu

1 Introduction

Increasing innovation among attackers, the increasing penetration of broadband Internet service and persistent vulnerabilities in host software systems have led to new classes of rapid and scalable mechanized attacks on the information infrastructure. Leveling the playing field requires scalable, automated responses to malicious code that can react in the short propagation windows evident with network worms such as Slammer [13]. Traditional approaches have relied on signatures, manual containment and quarantine (*e.g.*, [48]), and while tools are improving, reliance on identifying signatures and other improvements in detection processes is by itself insufficient. What is needed to complete the defensive technology portfolio is a scalable, distributed, adaptive response mechanism, based on cooperative behavior amongst a set of responding nodes. Since naïve cooperative behaviors might introduce new risks, including fragility in the face of poor or maliciously-generated information, particular attention must be paid to robustness in the cooperative strategy.

The problem of detecting, quarantining and recovering from zero-day viruses[1] is made easier if local detectors are allowed more room for error. If we err on the side of allowing false alarms, then detectors can be cautious (paranoid!) and conservatively flag anything that looks suspicious, and depend on cooperative corroboration to determine whether the attack is real or not. For this strategy to be effective, though, requires the entire anti-virus system to handle false alarms quickly and cheaply and still respond rapidly to real virus attacks.

Handling individual false alarms is not sufficient, however; by allowing more false alarms we increase the probability that the system will be called upon to manage multiple *potential* viral attacks taking place simultaneously. Simultaneous attacks complicate the anti-virus response be-

[1] In this paper we use the term "virus", in an extremely broad manner, to refer to any epidemic-like attack communicated over the network. We use this term, regardless of whether the virus is an active worm that attacks a system even without the unwitting involvement of a legitimate user or a passive virus, that is embedded in a document or email, that requires (unintentional) user assistance to become active.

cause increasing the defense against one virus involves either decreasing the defense against another virus or incurring higher costs (if the system can afford any further anti-virus costs). Simultaneous attacks may occur because of multiple zero-day viruses [36], or because early-stage false-alarms are not yet distinguishable from real virus attacks, or because of the resurgence of old viruses. Old viruses are still potentially virulent, since a measurable fraction of hosts do not upgrade or patch to eliminate security bugs, as the persistence of Code Red and other worms has demonstrated. A good analysis of the persistence of Blaster is given in [6], which shows that tens of thousands of instances of the virus remain active a full year after the initial outbreak. More surprisingly, 73%–85% of infected class-C subnets were not detected as infected during the first Blaster outbreak.

These observations expose several significant problems that must be dealt with. Any node that responds to a potential virus carries a cost: a node has finite resources and therefore can only actively engage a limited number of viruses at a time. Deciding to counter one virus entails ignoring some other virus. In the absence of cost, the best response to a potential virus attack is to flood the network as rapidly as possible with information to detect and counter the virus, causing as many cooperating agents to respond at once. The main question is simply whether the response is quick enough to stifle the virus. In the presence of a cost model, however, we still need to respond quickly, but no more quickly than necessary. A false alarm, whether malicious or unintended, can trigger a DoS attack by the response mechanism itself.

In this paper, we investigate tradeoffs in global, distributed response mechanisms that must respond quickly to real viruses and do not over-react to a high rate of false alarms. These systems should be efficient in terms of bandwidth and global computation. Moreover, the response mechanism must be robust against malicious agents spreading false information and be able to manage its resources even when many distinct viruses are active at any time. This approach is orthogonal to, and can augment, any proposals for the detection of and recovery from zero-day viruses. It has the added advantage of also performing well in the face of false alarms resulting from malicious behavior or failed detectors.

We focus here on an algorithm called COVERAGE (for COoperative Virus Response AlGorithm), whose core ideas were originally introduced in [3]. This algorithm takes into consideration shared information about the observed rate of infection for each virus, verifying that new reports are compatible with a node's own empirical observations, and determines (probabilistically) which viruses to respond to. We evaluate its effectiveness through large-scale simulation. We discuss also, although to a lesser extent, tradeoffs in a similar cooperative (but non-distributed) approach, called NRL03, described in [51], which differentiates between slow and fast-spreading viruses.

In our previous work, we determined that our basic cooperative and distributed approach was effective, but only in the sense of measuring the ability of the two approaches to detect and respond to viruses of different infection rates, as well as their resistance to malicious nodes that spread misinformation. Here, we offer refined algorithms over the earlier COVERAGE work, and, using a more detailed model we begin to examine the three-way tradeoff between communication costs, computation overhead, and the percentage of the network that gets infected before the reaction mechanism manages to limit the virus propagation. When evaluating alternative approaches in this model, we determine that COVERAGE always has much lower scanning costs; whether in the cases of slowly-propagating viruses, fast worms, or in the presence of malicious nodes injecting false alarms. COVERAGE has significantly lower communication cost when dealing with false alarms. Furthermore, the distributed approaches (both COVERAGE and our refined version) can trade off higher communication cost to to detect and react to slow-propagating viruses more quickly than the purely cooperative NRL03. Both NRL03 and the COVERAGE variants can use increased communication costs to perform better against fast worms, but the distributed approaches incur much higher communication costs than the NRL03 system. We believe this to be an unavoidable consequence of their robustness against false alarms. Independent work [9] has pointed out that the randomized approach of COVERAGE makes it difficult to devise virus propagation strategies that exploit the particular topology and exchange models of other collaborative virus defenses to hide their spread.

2 System Model

In this section, we describe our model of how viruses, switches/routers, hosts and our detection mechanism behave.

Modeling Viruses We use a fairly simple model to describe the behavior of potential attackers (viruses) that we consider in our work. After infecting a node, a virus attempts to infect other nodes; it may attempt to only infect a (small) fixed number of other nodes, or exhibit a greedier behavior. For our purposes, the distinction between the two types is simply in the probability of detection of a probe or attack by a detector. A virus may exhibit high locality of infection (*i.e.,* probing and attacking nodes based on network-topological criteria, such as "adjacent" IP addresses), or could use a random (or seemingly random) targeting mechanism, *e.g.,* using a large hit-list, or some pseudo-random sequence for picking the next address to attack. We expect that viruses that exhibit high locality are more difficult to detect using an Internet-wide distributed detection mechanism, but easier to do so on a local basis. We completely characterize a virus by the rate at which it attempts to infect other nodes and by the fraction of local attempts it makes. All attacks on susceptible nodes are successful, and in our simulation a virus never attempts to attack a non-existent node. As a result, our simulated viruses are more virulent than equally aggressive viruses in the real world. We make no assumptions about the infection vector: although perhaps the more "interesting" cases are those

where the virus is able to automatically subvert a machine or application, our model does not preclude human interaction in the infection process (*e.g.*, mail viruses as attachments).

Furthermore, we only assume that, once detected, there is some detection and/or response "module" associated with each virus — we do not investigate its details: the mechanism may be as simple as a content filter. There is some cost (in terms of CPU, memory, impact on legitimate communications, *etc.*) associated with each of these modules, which requires the prioritization of the various threats (viruses) in terms of allocating resources for detection and response.

Detection of Zero Day Worms and Viruses Although our algorithm is orthogonal to and agnostic about the method(s) with which new (zero day) viruses are detected, we briefly discuss different techniques and how they may interact with COVERAGE. A zero-day virus detector consists of roughly three components. First, we must detect anomalous behavior. The behavior may range from specific activities (*e.g.*, port scans, system/application crashes, incorrect password attempts) to statistical changes (*e.g.*, increased network traffic, slow response time, variation in system call signatures, number of TCP connections in TIME-WAIT). Second, the transmission vector must be identified (finding a set of network packets whose arrival seems to herald the onset of the anomalous behavior). Third, a detectable "signature" of the traffic must be generated so that hosts can scan for, and filter out, the potentially offending traffic. It is important to note that a "signature" in our model is not necessarily simply a pattern of bits to match inside a packet — it can be any profile that detects anomalous behavior, ranging from packet inspection to longer term multi-packet behavior.

Perhaps the most promising approach is that of monitoring the number of packets aimed at the unused portion of an organization's address space, as was suggested in [29]. In that work, it was shown that with as few as 4 such probes, it is possible to infer the existence of a new virus aimed at a previously untargeted service/port. A similar approach is proposed in [81], where sudden changes in the traffic statistics maintained on a per source IP address and per destination port number indicate a high-visibility event, such as a scanning worm. Similar works have proposed measuring the entropy of traffic (*e.g.*, in terms of distinct source IP addresses seen) as an indication of unusual activity. These mechanisms act as early warnings, alerting administrators and perhaps automatically reconfiguring a firewall to assume a more defensive posture. However, without corroboration with outside sources (*e.g.*, through COVERAGE) they can be manipulated by an attacker to generate false positive reports. It is also worth noting that these mechanisms can only give a rough fingerprint of a new virus's activity, such as the targeted service/port—thus, they can be fairly accurate about the presence of an attack, but inaccurate about mapping specific packets to the attack, as would be the case with a virus targeting a protocol such as HTTP.

A second, more accurate but also more expensive (computationally, as well as in terms of necessary infrastructure) mechanism for detecting viruses is through use of properly instrumented honeypots or virtual machines, as is done in [62, 35], or through payload analysis [32, 63] that can yield a potential virus signature. Finally, anomaly detection techniques, such as those proposed in [7], can indicate the presence of packet payloads that do not conform to the typical contents of packets for a particular service (*e.g.*, binary content containing a buffer overflow payload uploaded to a web server).

These mechanisms identify different points in the zero-day virus detection space, trading off between the likelihood of false positives, the time needed to collect enough evidence before raising an alarm, and the expense of testing whether an alarm should go off.

These observations are taken into consideration by the COVERAGE algorithm to balance the cost of detection (*e.g.*, coordination, scanning as well as collateral damage that may be caused by false alarms) and the ability to respond effectively to virus attacks.

Network Topology Our simulation topology is dictated by our assumptions about the vulnerabilities and capabilities of network nodes with respect to virus attacks. We assume that, as a general rule, routers/switches are less likely to be infected by a virus, and thus that only hosts are susceptible to infection.

Here, we assume that the only nodes in our system capable of scanning packet sequences for potential viruses are end-hosts or last-hop routers. While considerable advantage can be gained by exploiting the great levels of traffic aggregation seen in routers closer to the network core, it is unlikely that such nodes can actively scan for viruses without significantly affecting their performance.

Thus, our model of the network topology consists entirely of a collection of *subnets* (LANs) containing a number of *hosts*. Each subnet connects to the global network through a single *router*. All routers are connected together in a single cloud where each router can address and forward packets to each other directly. End-hosts can only see their traffic, while routers can inspect all traffic to or from their associated LAN. It is likely that some organizations contain multiple subnets that frequently communicate among themselves. Therefore we collect together several subnets into a *domain*. A domain captures particular communication patterns but has no structural impact on the topology for simulation.

State of Nodes We assume that the distribution of COVERAGE agents is uniform across the population of nodes; for example, all nodes may be running an agent, in the same way that a large number of PCs run some kind of anti-virus software these days. A node in our environment can be in one of three states with respect to a virus: *susceptible*, *protected*, or *immune*. A susceptible node can be either infected or uninfected. Susceptible nodes will become infected if subjected to an attack. Protected nodes may be infected or uninfected, but only if the detection module does not have the ability to

detect and disinfect an infected machine. A protected node will not become infected as long as the protection mechanism (typically, a module that screens packets or email) is in place. An immune node does not have the vulnerability exploited by the virus.

Operations A COVERAGE agent can monitor traffic and, for each virus, it can either ignore the virus or perform one or more of the following operations: collect and exchange information about a virus, *scan* for the presence of a virus (actually, scan for the presence of patterns of network traffic used as a "signature" for that virus), or *filter* viruses (by dropping one or more packets that are part of a virus signature). We assume that there is a cost inherent in checking for virus signatures. That is, a node cannot be actively "on the lookout" for an arbitrary number of viruses without adversely affecting its performance. (Some experimental measurements of such real-world limits are given in [4]). Edge-routers are more likely to be constrained by high packet rates, and therefore limited in the amount of scanning they can perform. Hosts can afford to scan for more viruses without interfering with their (lower) packet rate, but, on the other hand, have work other than packet forwarding to perform. In either case there is an upper bound on the number of viruses a node can scan for.

We assume that nodes periodically exchange information about viral infections. Although the per-virus cost of such an exchange is low, we assume that the number of known *plus potential day-zero* viruses exceeds the amount of information that can be reasonably exchanged at any given time. Thus, actively exchanging information about a virus incurs a cost, albeit lower than scanning. Note that faulty or otherwise malicious nodes (*e.g.,* nodes controlled by the virus) may lie in the information they provide as part of these exchanges.

Routers can additionally scan for suspicious behavior on all traffic to or from their LAN (and drop when necessary). We further assume that if a router detects a rampant viral infection for a virus that has an associated disinfectant component, the router can invoke a disinfection operation (perhaps alerting an administrator) on all the nodes in its LAN.

Model of Anti-virus Epidemic Each node participating in the anti-virus response must make certain decisions: (a) the rate at which it polls other local nodes for virus information, (b) the rate at which it polls other remote nodes, chosen at random, for virus information, (c) whether for each virus to collect information about it, (d), whether to include that information in virus exchange packets, and (e) whether to scan for the virus (collecting the results of those scans as part of the local information for that virus).

3 Cooperative Virus Response

COVERAGE tries to balance the cost of scanning and filtering packets for a specific virus against the benefit of detecting, other, real viruses in several ways. First, COVERAGE models the virulence of viruses and ranks them in virulence order. With probability proportional to their virulence, COVERAGE decides in rank order whether to actively scan for the virus or not. COVERAGE stops scanning for more viruses once the scanning schedule consumes the entire scanning budget available. Second, each COVERAGE agent exchanges information about the state of a virus with other cooperating agents in order to construct a model of the virus and determine whether incoming reports are empirically consistent with the observed state of the network. Third, COVERAGE agents determine their polling rate to maximize the probability of seeing enough viruses to confirm the current local estimate of the virus state, while reducing the probability that communication will add no new knowledge to either of the participants. We now describe the algorithm in more detail.

3.1 COVERAGE algorithm

Agent communication. Each COVERAGE agent polls other agents, selected randomly. Assuming that only a small fraction of the nodes are reporting false information, a randomly selected node is more likely to be trustworthy than a node that actively contacts us — a small number of malicious nodes may try to flood the rest of the network. At each poll, the sender reads the response and updates its local state variables to track the operation of the cooperative response mechanism and the status of the network in terms of observed attacks.

First, it records whether the remote agent is actively scanning. This allows the agent to estimate the fraction of agents in the network that are actively scanning for a particular virus. Second, it updates estimates of possible infections *e.g.,* the fraction of infected nodes for each virus. We distinguish two types of estimates: direct and remote. Direct estimates are updated based on whether each remote agent has directly observed an attack (either to itself or, if a router, to a node in its LAN). Remote estimates are updated based on the fraction of infected nodes as estimated by the remote agent (the "direct" estimates of the remote agent). Direct measurements performed by the local node are absolutely trustworthy — there is no issue of false positives. The direct measurements of agents that we poll (which become our remote estimates) are next in trustworthiness. Remote estimates of agents who we poll are more suspect, and information reported by agents who contact us are the most suspicious of all. However, we can validate any information reported to us — if someone reports that a particular virus is attacking 25% of the Internet at the moment, then if we poll 20 agents at random (and assuming uniform distribution of COVERAGE agents across the node population), then with 80% probability we would expect to find that between 3 and 7 of those agents had directly seen an attack in the last measurement interval. Values outside that range would cast doubt on the remote estimate.

Finally, in this paper we ignore the details of how COVERAGE nodes authenticate themselves to each other. How-

ever, we note that even strong authentication is not sufficient for our system. If a COVERAGE agent is taken over by a malicious attacker, then the attacker can (presumably) still authenticate itself and discover which nodes are *not* scanning for a particular virus, and use that information when choosing targets. To defend against such a vulnerability in COVERAGE , we propose (but have not yet implemented or experimented with) a simple defense. When polling, the identity of the target agent is not important — just the fact that we chose it randomly, and it did not choose us. And, while we are interested in the statistics of the sample as a whole, we need not link a particular set of direct measurements to a particular IP address. Consequently, each agent stores a randomly selected response from the last measurement interval (the local measurements are one of the candidates that may be selected), and returns that random selection in response to any COVERAGE poll, for the direct measurements and scan list only. (The cumulative counters are still stored and reported accurately). The poller still receives an accurate response — just perhaps from a different IP address than the one it polled, and perhaps slightly older than expected. This adds a level of indirection to the polling process.

Periodic updates. At regular intervals each COVERAGE agent updates its state based on the information received since the last update. To track the progress of the infection each COVERAGE agent maintains a smoothed history for each type of estimate (direct and remote), each as exponentially decaying averages with varying time constants, to approximate recent infection rate, past rate, and background rate.

Using these estimates, an agent can compute the fraction of nodes believed to be infected as well as the growth of the infection, assuming exponential growth[2].

If we assume that each infected node infects roughly α nodes in a given timestep, and that a fraction $p^\star$ of all nodes are infected at present, t_0, then we'd expect that at t timesteps in the past, we would have seen a fraction $p^\star(1+\alpha)^{-t}$ infected nodes in our sample. For each virus we are observing, we can fit our observations at timestep $t_0 - t$ to a growth curve $p^\star(1+\alpha)^{-t}$. We can use a least square fit to find the best values of $p^\star$ and α.

In practice, however, at the early stages of the virus, the fraction of infected nodes in a sample will jump wildly. The pattern will only emerge after a relatively large number of timesteps. Rather than recording a large number of samples, and expensively curve fitting the full set of observations, we instead use exponentially decaying averages with different time constants. Each observed fraction, f, is incorporated into the jth average a_j, by $a_j = (1-w_j)a_j + w_j f$. If the current fraction of infected nodes is $p^\star$, and the process were exactly exponential, then at time t_0 we'd expect each ideal $a_j^\star$ to contain:

$$a_j^\star = \sum_{i=1}^{\infty}(1-w_j)^{i-1} w_j \frac{p^\star}{(1+\alpha)^{i-1}}$$

$$a_j^\star = w_j p^\star \sum_{i=1}^{\infty}\left(\frac{(1-w_j)^{i-1}}{(1+\alpha)^{i-1}}\right)$$

$$a_j^\star = w_j p^\star \sum_{i=0}^{\infty}\left(\frac{(1-w_j)}{1+\alpha}\right)^{i}$$

$$a_j^\star = w_j p^\star \frac{1}{1-\frac{(1-w_j)}{1+\alpha}}$$

$$a_j^\star = w_j p^\star \frac{1+\alpha}{w_j+\alpha}$$

If we maintain three decaying averages, one with a large w_j to capture recent history, one with a small w_j to get the background level, and one with an intermediate w_j, then we can compute the mean square error between our actual observed a_j and the expected $a_j^\star$ assuming that growth was exponential with base α and current population $p^\star$ of the network. We can then choose the α and $p^\star$ that minimize the mean square error between our observations and the expected values of a_j.

We start with the computation of the mean square error

$$\text{MSE} = \sum_{j=1}^{3}\left(w_j p^\star \frac{1+\alpha}{\alpha+w_j} - a_j\right)^2$$

Grouping the equation for MSE by $p^\star$,

$$(p^\star)^2\left((1+\alpha)^2\sum_{j=1}^{3}\left(\frac{w_j}{\alpha+w_j}\right)^2\right) - p^\star\left(2(1+\alpha)\sum_{j=1}^{3}\frac{w_j a_j}{\alpha+w_j}\right) + \sum_{j=1}^{3}a_j^2$$

we get the derivative,

$$\text{MSE}' = 2(1+\alpha)^2 p^\star \sum_{j=1}^{3}\left(\frac{w_j}{\alpha+w_j}\right)^2 - 2(1+\alpha)\sum_{j=1}^{3}\frac{w_j a_j}{\alpha+w_j}$$

Setting the derivative equal to 0, we find the ideal $p^\star$ in terms of α

$$p^\star = \frac{2(1+\alpha)\sum_{j=1}^{3}\frac{w_j a_j}{\alpha+w_j}}{2(1+\alpha)^2\sum_{j=1}^{3}\left(\frac{w_j}{\alpha+w_j}\right)^2}$$

$$p^\star = \frac{\sum_{j=1}^{3}\frac{w_j a_j}{\alpha+w_j}}{(1+\alpha)\sum_{j=1}^{3}\left(\frac{w_j}{\alpha+w_j}\right)^2}$$

Fig. 1 Deriving $p^\star$ from α to minimize MSE between observed a_j's and expected values of a_j's assuming exponential growth.

[2] We assume all growth is exponential for the purpose of deciding whether to trigger a reaction. We believe that linear-growth viruses can be detected by humans, and need not be countered by an automatic, distributed, algorithm. If our assumption is incorrect and growth is, in practice, sub-exponential then we recover naturally because we observe a decrease in α and gradually back-off as the predicted "virulence" of the virus drops.

We have

$$p^\star = \frac{\sum_{j=1}^{3} \frac{w_j a_j}{\alpha + w_j}}{(1+\alpha)\sum_{j=1}^{3}\left(\frac{w_j}{w_j+\alpha}\right)^2}$$

Plugging this back into the equation for MSE, we get:

$$\text{MSE} = \left(\frac{(1+\alpha)\sum_{j=1}^{3}\frac{w_j a_j}{\alpha+w_j}}{(1+\alpha)\sum_{j=1}^{3}\left(\frac{w_j}{w_j+\alpha}\right)^2}\right)^2 \left(\sum_{j=1}^{3}\left(\frac{w_j}{\alpha+w_j}\right)^2\right) - \frac{2(1+\alpha)\sum_{j=1}^{3}\frac{w_j a_j}{\alpha+w_j}}{(1+\alpha)\sum_{j=1}^{3}\left(\frac{w_j}{\alpha+w_j}\right)^2}\left(\sum_{j=1}^{3}\frac{w_j a_j}{\alpha+w_j}\right) + \sum_{j=1}^{3} a_j^2$$

$$\text{MSE} = \frac{\left(\sum_{j=1}^{3}\frac{w_j a_j}{\alpha+w_j}\right)^2}{\sum_{j=1}^{3}\left(\frac{w_j}{\alpha+w_j}\right)^2} - \frac{2\sum_{j=1}^{3}\frac{w_j a_j}{\alpha+w_j}}{\sum_{j=1}^{3}\left(\frac{w_j}{\alpha+w_j}\right)^2}\sum_{j=1}^{3}\frac{w_j a_j}{\alpha+w_j} + \sum_{j=1}^{3} a_j^2$$

$$\text{MSE} = \frac{\left(\sum_{j=1}^{3}\frac{w_j a_j}{\alpha+w_j}\right)^2 - 2\left(\sum_{j=1}^{3}\frac{w_j a_j}{\alpha+w_j}\right)^2}{\sum_{j=1}^{3}\left(\frac{w_j}{\alpha+w_j}\right)^2} + \sum_{j=1}^{3} a_j^2$$

$$\text{MSE} = \sum_{j=1}^{3} a_j^2 - \frac{\left(\sum_{j=1}^{3}\frac{w_j a_j}{\alpha+w_j}\right)^2}{\sum_{j=1}^{3}\left(\frac{w_j}{\alpha+w_j}\right)^2}$$

Fig. 2 Equations to derive value of α that minimizes MSE between observed a_j's and expected values of a_j's assuming exponential growth.

The mean square error is (we drop the constant factor of 1/3 for clarity)

$$\text{MSE} = \sum_{j=1}^{3}\left(w_j p^\star \frac{1+\alpha}{w_j+\alpha} - a_j\right)^2$$

Note that given a fixed α we can easily compute the $p^\star$ that minimizes the MSE, as this is a quadratic equation in $p^\star$. Figure 1 gives this derivation, and we see that

$$p^\star = \frac{\sum_{j=1}^{3}\frac{w_j a_j}{\alpha+w_j}}{(1+\alpha)\sum_{j=1}^{3}\left(\frac{w_j}{\alpha+w_j}\right)^2}$$

At this point, we have two alternative numerical techniques for computing α. First, we can plug our value of $p^\star$ back into the equation for the MSE, and find the value of α that minimizes the MSE (see Figure 2). Second, alternatively, we can (numerically) find the optimal α for a given $p^\star$, and then analytically find the optimal $p^\star$ for the new α, and iterate until the MSE stops decreasing. In theory these should be equivalent; however the second approach seems to work better in practice (mostly because it is easier to avoid local minima, and also to stay within regions where the computation converges).

Given estimates of $p_d^\star$ and α_d, we can calculate the *virulence*, v_d, of a virus as the estimated number of timesteps needed by the virus to infect the entire network. We can solve for the number of steps, v_d, it takes for $p_d^\star(1+\alpha_d)^{v_d} = 1$.

$$p_d^\star(1+\alpha_d)^{v_d} = 1.\log(p_d^\star) + v_d\log(1+\alpha_d) = 0$$
$$v_d\log(1+\alpha_d) = -\log(p_d^\star)$$
$$v_d = \frac{-\log(p_d^\star)}{\log(1+\alpha_d)}$$

Note that we independently calculate virulence for global and local growth, in order to identify attacks that are non-uniformly distributed throughout the network. Using the same method as above the agent also computes α_r, $p_r^\star$ and v_r based on the remote estimates.

Scanning/filtering. Given the estimates an agent can decide whether it needs to scan for a given virus. There is a basic, low level of scanning for every virus. When a virus becomes active the scanning rate may increase. In the general case, the agent can sort viruses in order of their virulence v_d and decide whether to scan for each virus, in turn, stopping when the scanning budget is filled. (In our simulation, we only scan viruses whose v_d is below *threshold*.)

To maintain a basic, low level of scanning for every virus, every agent measures the fraction $f_{scanning}$ of nodes in the network that are actively scanning for a given virus based on information exchanged with other nodes. If this fraction is below a threshold f_{target} (around 2-5%) and the node has enough resources for scanning, it activates with probability $f_{target} - f_{scanning}$, and disables scanning in a similar way if too many nodes seem to be active. To avoid turning "blind" to certain viruses because of a fraction of malicious nodes falsely reporting that they are actively scanning, nodes need to aim for $f_{target} + f_{malicious}$, where $f_{malicious}$ is the maximum tolerable fraction of malicious nodes. Although this increases scanning cost, nodes can trade-off this cost for higher communication costs.

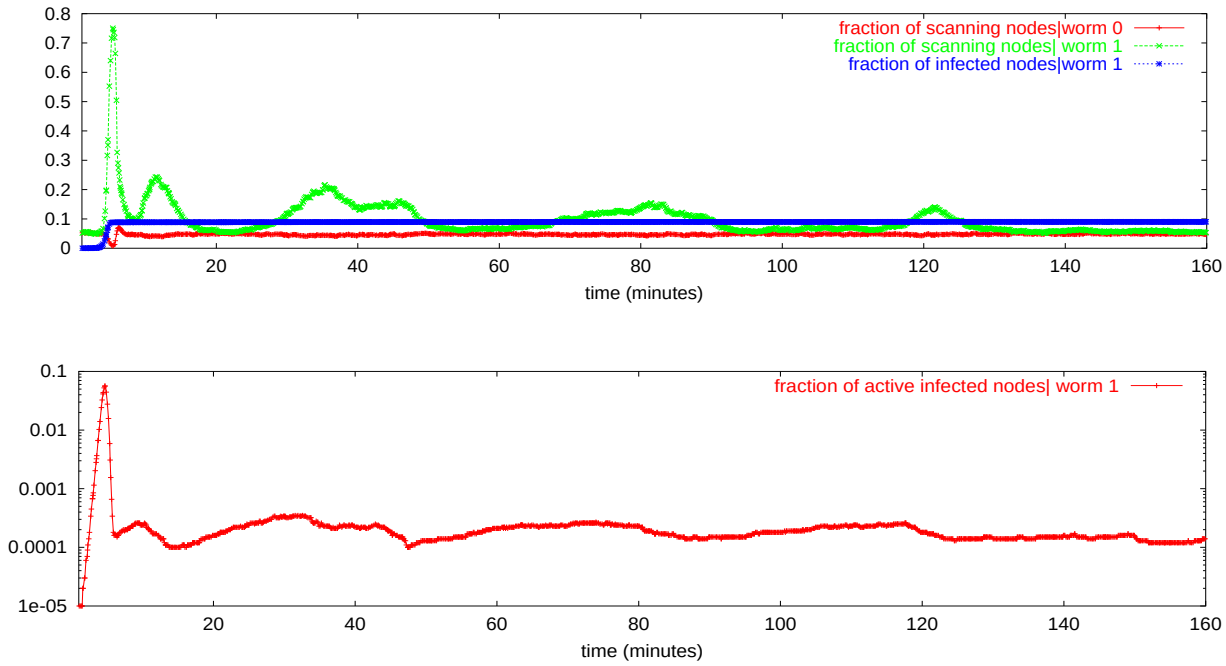

Fig. 3 Fractions of infected hosts and scanning nodes over time (top), and fraction of actively infected nodes (bottom).

An inactive agent, A, may also start scanning seemingly low-virulence viruses, if *enough other* agents claim the virus is virulent, and A finds that the fraction of scanning nodes is too low to detect virus activity in a single timestep at the current polling rate. The test is whether n (simply the fraction of agents that were polled and found to be scanning in the last interval) is less than twice the estimated fraction of infected hosts (*e.g.*, if $n < 2p_r^\star$). Similarly, if the agent is active but $n > p_r^\star$ then it decides to stop. The agent also stops scanning if α_r approaches 0. This ensures that the fraction of scanning agents is bounded if there is insignificant progress for a given infection or if the infection is small compared to the number of actively scanning agents. Such heuristics are essential for controlling the behavior of the algorithm, keeping the response mechanism "ahead" of the virus but also limiting the damage and cost when malicious agents spread false information.

A small number of agents need to be watching for each dormant virus. The number of active scanners monitoring a virus may be more than warranted by the level of virus activity. An agent detecting this will stop monitoring the virus. If the agent finds that it now has ample room within its scanning budget to consider another virus, it chooses another virus to monitor uniformly at random from the (large) virus database. The agent may choose a virus that almost no one else is scanning for — in which case it will stay on the scanning list for a long time, and be inspected by the agent as long as there are not too many virulent virii. If the new virus is dormant, but enough people are already looking at it, then the agent will drop it, and randomly choose another.

Polling rate. An agent communicates with other agents within the same domain at a constant, high rate, as the cost of intra-domain communication is assumed to be very small. Inter-domain communication is generally more expensive; agents therefore need to adapt the rate of polling remote agents, avoiding excessive communication unless necessary for countering an attack. When there is no virus activity, agents poll at a pre-configured minimum rate (at least an order of magnitude lower than the rate for intra-domain communication). An agent periodically adapts the remote polling rate if v_r is less than a given threshold. The new rate is set so that the agent polls $1/(p_r^\star)^2$ remote agents in each update interval, unless this rate exceeds a pre-configured maximum rate. This is used to increase the polling rate when the remote estimate indicates that an attack is imminent (but not yet reflected in the direct estimate). If the more recent direct estimate $p_d[n]$ is non-zero, then the polling rate is increased so that at least a few samples can be collected in each update interval. Finally, if the estimated virus population $p_r^\star$ is small and the estimated virus growth rate is close to zero, the agent throttles back its remote polling rate to the minimum rate.

These adjustments are always performed on the polling side. We avoid changing the state or behavior of the polled agent to reduce the risks associated with malicious agents. Otherwise, they could spread misinformation and raise false alarms more effectively by increasing their own communication rate.

3.2 COVERAGE behavior

To give a rough sense of how the COVERAGE algorithms described above behave, Figure 3 displays a single example run of the COVERAGE algorithm against a single simulated

virus called "worm 1". We show the activity of the virus (the number of nodes that were ever infected in their lifetime) on the top figure, and the currently infected nodes on the bottom figure), as well as the response of COVERAGE (both the number of agents scanning for "worm 1", as well as the number of agents scanning for a dormant virus "worm 0"). (Section 4 describes how we approximate a heavy load on the COVERAGE agents by using a simulation parameter `threshold` — each agent is too busy to consider any viruses unless they are likely to take over the entire network within `threshold` measurement intervals.)

One can see the initial stage of the infection and the response of the algorithm: the virus manages to infect roughly 10% of the hosts; cooperation between COVERAGE agents results in a rapid activation of filtering on roughly 75% of the network effectively eliminating the virus. Soon after stopping the attack, the COVERAGE agents on uninfected parts of the network deactivate scanning/filtering. However, Figure 3 (bottom) shows, a small number of hosts remains infected and undiscovered, resulting in another three episodes where COVERAGE agents are activated (each episode with a smaller fraction of agents activated) to defend against a secondary outbreak. Although a tiny fraction of infected nodes remains undiscovered, it does not cause any further harm and COVERAGE gives users time to patch up their systems. The scanning for dormant "worm 0" continues, except during the most virulent part of the outbreak, where the number dips as resources are marshaled to defend against "worm 1".

4 Simulation Results

To simplify the analysis of COVERAGE and to meaningfully include the non-adaptive NRL03, we restrict the simulation to a single virus. We model the impact of multiple active viruses by assuming that each node is already very busy handling other viruses. To represent the load imposed by other viruses, we specify a threshold under which a virus will not have high enough priority to be scheduled in the scanning budget. If many viruses are active then the threshold will be a small number, such as 20. (Recall that the virulence is a measure of how many measurement intervals it will take before the virus has covered the *entire* Internet. Normally, when the network is not under any attack, then a node is likely to scan or filter a virus, even if its virulence is only 100 or 1000.) Unless the current virus is poised to conquer the entire net at its current rate of growth from its current coverage within `threshold` intervals, it will not have high enough priority to be scheduled in the scanning budget. In our simulations, we only consider cases where the net is already under such a heavy attack that scanning nodes ignore any virii that are not poised to control the entire network within small thresholds of 5 or 20. (These correspond to times from 10 seconds to 2 minutes).

To better understand the performance of COVERAGE, we limit our simulation to a simple, relatively small network of 100,000 edge-routers, each connected to 8 hosts, with 50 edge-routers in each of 2,000 domains. We also consider the performance of our version of COVERAGE in relation to NRL03 [51], another cooperative algorithm, which makes different tradeoffs than COVERAGE. NRL03 uses cooperative peer-to-peer strategies to respond to large-scale Internet virus attacks. The model involves a number of *friend* nodes, which work together by exchanging information to warn of suspicious virus-like network behavior. The larger the number of friends, the more rapidly NRL can respond to virus attacks — and the higher the communication costs. As in COVERAGE, a small fraction of nodes is assumed to be scanning for a given virus. When the virus is detected, the node broadcasts the alert to its friends. When a node receives such an alert, it increments an alert counter, and propagates the alert to its set of friends when this counter reaches a threshold.

For the COVERAGE algorithm, we set the local-domain polling interval to 1.8 seconds , the maximum and minimum remote polling intervals to 6 seconds and 1.8 seconds respectively. For both algorithms we assume that 4% of the edge-routers are permanently scanning for the virus.

Our analysis uses three metrics. First, we model the success of the attack by integrating the number of infected nodes over time. This is only relevant in the case of a real virus attack. Second, we consider the number of edge-routers actively scanning/filtering this virus. This is a measure of the computational overhead of the response mechanism. Our third metric, the total number of messages sent, measures the communication cost.

We measure the progress of infections of differing virulence and the success of the response mechanism as the integral over time of the fraction of infected nodes. Because we claim that COVERAGE is better able to balance multiple viral attacks and NRL03 makes no such claim, we conservatively model COVERAGE dealing with other viral outbreaks, but optimistically let NRL03 assume that this is the only virus in the Internet. The results for COVERAGE and NRL03 with different parameters are shown in Figure 4. (It may seem counter-intuitive that the more virulent viruses cover less of the network; however, recall that we are integrating over time and that the faster worms, while they spread faster are also detected and disinfected faster). COVERAGE reacts more slowly than NRL03 for fast worms in our setting — where NRL03 handles a single virus, and COVERAGE faces many aggressive virii. Consequently, the virus takes a larger initial toe-hold in the network under COVERAGE, and is active slightly longer before being cleaned up. Because of this toe-hold, COVERAGE performs relatively worse in our setting than NRL03 for fast worms. On the other hand, even when handicapped under our model it still detects the slow viruses before NRL03. The slower response by COVERAGE in the case of fast worms has been a deliberate design choice in an attempt to make the algorithm robust against false information from potentially malicious nodes.

The default configuration of COVERAGE varies the communication rate between roughly .016 rounds of messages

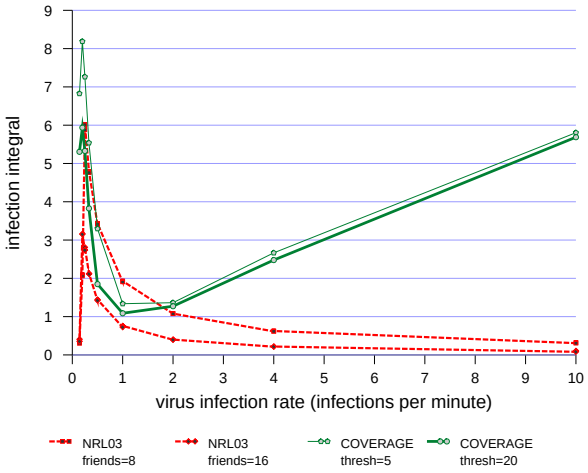

Fig. 4 Integral of infected hosts over time *vs.* virus infection rate. The integral is divided by the elapsed time and expressed as a percentage. (100 implies that all nodes were infected all of the time.)

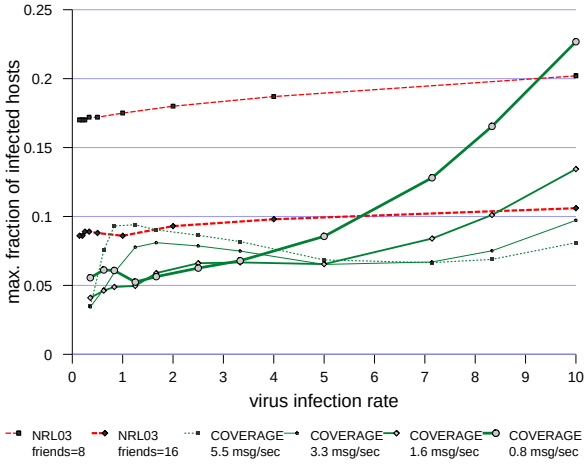

Fig. 5 Maximum fraction of infected hosts *vs.* virus infection rate when under attack.

per second and roughly .055 per second. (In each round a COVERAGEnode will vary the number of remote notes it sends to based on the current estimate of virulence and coverage. The total number of messages will be the number of rounds per second times the average number of remote nodes it contacts.) Figure 5 shows the effect of increasing the maximum allowable rate that COVERAGE communicates with remote nodes. We plot the high water mark of viral attacks as a function of the virus infection rate for different maximum communication rates in COVERAGE. We can see that a moderate increase in communication rates[3] in COVERAGE allows it to stop the virus with a lower high water mark than NRL — even with friends = 16. (Figure 8 shows that, for highly active worms, the default configuration of COVERAGE has communication costs comparable (within 20%) to NRL with friends=8, and half the communication costs of NRL with friends = 16.)

[3] The communication costs for COVERAGE scale with the virulence and number of active viruses, and are thus more scalable than NRL — still, we are investigating ways of conveying the necessary polling information more efficiently during quiet periods.

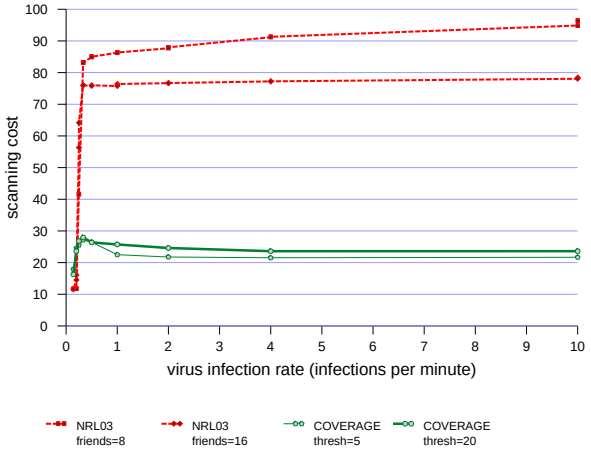

Fig. 6 Scanning activity (percentage of scanning nodes) *vs.* virus infection rate when under attack.

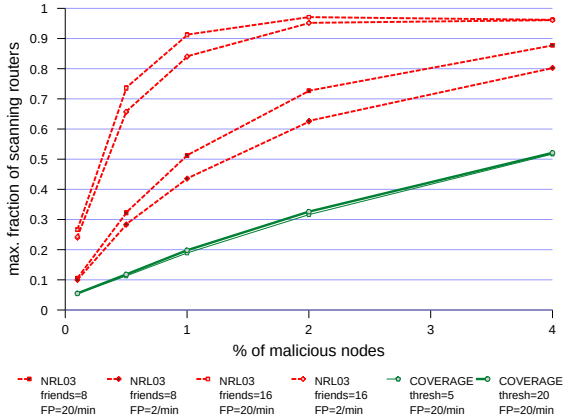

Fig. 7 Maximum fraction of false scanning routers with malicious nodes and false alarms. We consider two kinds of malicious nodes: one that generates 2 false positives per minute, and one that generates 20 per minute.

Figures 6 and 7 show the fraction of nodes scanning for a virus as a function of the virulence of the virus and the fraction of malicious (or faulty) nodes. Malicious nodes may be simply faulty, or may be controlled by an adversary (*e.g.*, infected nodes). Such nodes may then be spreading false information, in an attempt to misdirect COVERAGE. Possible goals include attempting to hide a virus outbreak, or causing COVERAGE to waste resources. Alternatively, (perhaps more commonly), the "malicious nodes" may simply be virus detection mechanisms that generate false positives. The figures for COVERAGE are more pessimistic than for NRL, because in NRL *every* edge-router is scanning for the virus (obviously impractical for a large set of viruses), and the graphs report only those nodes that are actively *filtering* the virus. The COVERAGE plots report the fraction of nodes that even scan for the virus at all. A smaller number (unreported here) are filtering for the virus. Nevertheless,

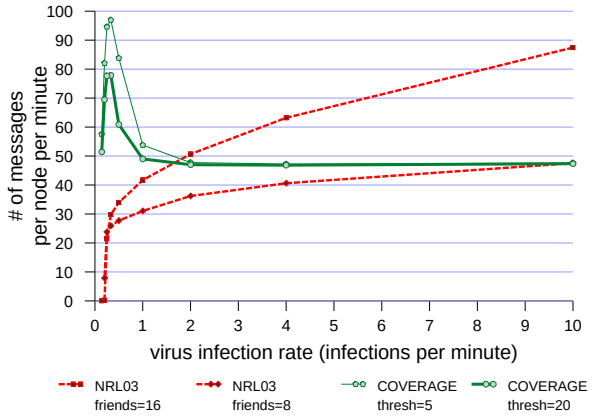

Fig. 8 Communication cost *vs.* virus infection rate when under attack.

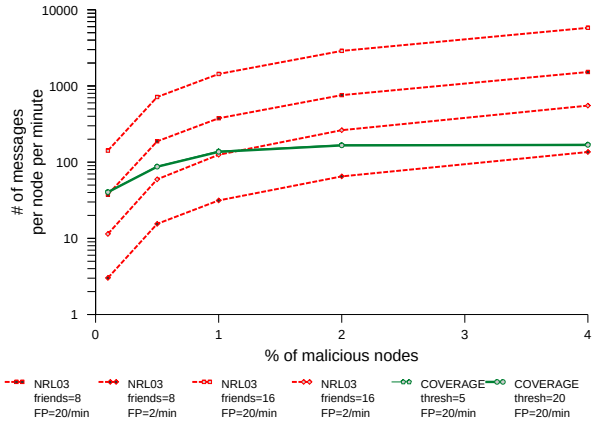

Fig. 9 The impact of malicious routers and false alarms on communication cost.

the pessimistic ("worst-case") results for COVERAGE represent far lower scanning costs than the optimistic ("best-case") results for NRL03. (Even reducing the malice (i.e. dropping the false positive rate by a factor of 10, down to 2-per-minute) of the malicious nodes can still trigger filtering in a large fraction of the NRL03 nodes). COVERAGE manages to control the virus with a much smaller set of scanning nodes, and it similarly detects false alarms with fewer nodes triggered to scan or filter.

Figures 8 and 9 demonstrate that the communication costs for COVERAGE in the face of false alarms is much lower than for NRL03 — understandably because COVERAGE correctly identifies the suspicious behavior as *false* alarms. For slow-growth viruses, COVERAGE requires significantly more communication to convince cooperating peers that a virus attack *is* underway. However, this extra cost conveys a benefit: COVERAGE detects slow-growth viruses long before NRL03 is able to. For fast worms, communication costs are generally comparable — NRL requires considerably more communication when Friends = 16, but it should be noted that NRL03 controls the infection more rapidly than COVERAGE in these cases. For COVERAGE to control fast worms as effectively as NRL03 would require even higher communication costs.

The impact of false alarms on performance in detecting a real attack is illustrated more clearly in Figure 10. The false alarms and reports from malicious nodes confuse COVERAGE's picture of the real attack. Each line in the graph represents a different level of virulence of a virus attack. We see that, for each of the lines, the high water mark of the attack grows almost linearly with the number of malicious nodes. Roughly double the number of nodes are infected when 4% of the nodes are malicious compared to a system without any malicious nodes. Roughly twice, again, as many are infected when 10% (more than double) the nodes are malicious. We see that until the per-node infection rate approaches 5 infections per minute, the number of infected nodes is roughly comparable to the number of malicious nodes. Each malicious node, then, allows on average only a single additional

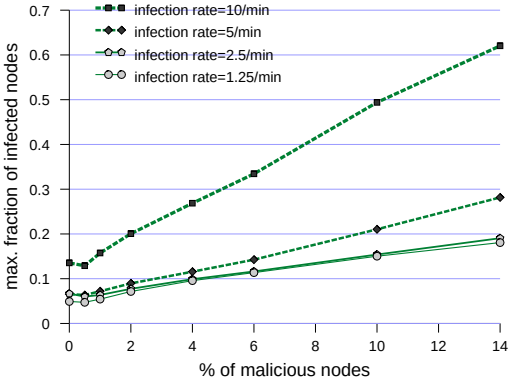

Fig. 10 Impact of false alarms and malicious nodes on detection performance.

node to get infected by the real virus. Malicious nodes can thus do very little to distract COVERAGE from fighting the real attack. Even under more aggressive attacks, it takes a large number of malicious nodes reporting false alarms in order to distract COVERAGE in a significant manner.

5 Related Work

Computer viruses have been studied extensively over the last several decades. Cohen was the first to define and describe computer viruses in their present form. In early work [18], he gave a theoretical basis for the spread of computer viruses.

The strong analogy between biological and computer viruses [28] led Kephart *et al.* [31] to investigate the propagation of computer viruses based on epidemiological models. They extend the standard epidemiological model by placing it on a directed graph, and use a combination of analysis and simulation to study its behavior. They conclude that if the rate at which defense mechanisms detect and remove viruses is sufficiently high, relative to the rate at which viruses spread, they can prevent widespread virus propagation.

Miretskiy *et al.* [46] describe a filesystem layer designed specifically for efficient virus scanning and removal.

In [68], the authors describe the risk to the Internet due to the ability of attackers to quickly gain control of vast numbers of hosts. They argue that controlling a million hosts can have catastrophic results because of the potential to launch distributed denial of service (DDoS) attacks and access any sensitive information that is present on those hosts. Their analysis shows how quickly attackers can compromise hosts using "dumb" worms and how "better" worms can spread even faster. They also envision a Cyber "Center for Disease Control" (CCDC) for identifying outbreaks, rapidly analyzing pathogens, fighting the infection, and proactively devising methods of detecting and resisting future attacks. Our work focuses on the strategies of distribution and deployment of detectors and antidotes produced by such a CCDC. In subsequent work [67], the same authors show how a worm using pre-compiled lists of IP addresses known to be vulnerable can infect one million hosts in half a second. Antonatos *et al.* [5] propose using address randomization through aggressive use of DHCP to make the hitlist information stale.

Chen and Ji show that reasonable estimates of the vulnerable application's distribution across the network can be made dynamically by a worm, which can adjust its spreading parameters to optimize infection rate [15]. Other work has shown that worms can make detection of compromised hosts harder by using simple, distributed but coordination-free mechanisms that stop their spreading (turning infected hosts into "sleeper agents") once the epidemic has reached its apex [42].

Leavitt [34] discusses the threat of worms aimed at mobile phones, describing some of the first malware of this type. Zhou *et al.* [87] discuss worms that spread over peer-to-peer networks, exploiting the richer (and arbitrary) topologies to achieve accurate targetting and fast propagation. Mannan and van Oorschot [44] describe worms spreading over instant-messaging networks, and propose user throttling and disconnecting users with the largest contact lists as ways of controling such epidemics.

The CodeRed worm [12] was analyzed extensively in [89]. The authors of that work conclude that even though epidemic models can be used to study the behavior of Internet worms, they are not accurate enough because they cannot capture some specific properties of the environment these operate in: the effect of human countermeasures against worm spreading (*i.e.,* patching, filtering, disconnecting, *etc.*), and the slowing down of the worm infection rate due to the worm's impact on Internet traffic and infrastructure. They derive a new general Internet worm model called *two-factor worm* model, which they then validate in simulations that match the observed CodeRed data available to them. Their analysis seems to be supported by the data on CodeRed propagation by Moore *et al.* [47] and by Song *et al.* [65] (the latter distinguished between different worms that were active simultaneously). Similar analyses on the SQL "Slammer" (or Sapphire) worm [13] and for Witty worm are available [64, 61]. More recent analyses [88] show that it is possible to predict the overall vulnerable population size using Kalman filters early in the propagation cycle of a worm, allowing for detection of a fast-spreading worm when only 1% or 2% of vulnerable computers on the network have been infected. [6] shows that worms are surprisingly persistent, showing that Blaster remains active on tens of thousands of PCs a year after the initial outbreak. An analysis of Blaster and SoBig network traces is given by Dubendorfer *et al.* [25].

CodeRed inspired several countermeasure technologies, such as La Brea [39], which attempts to slow the growth of TCP-based worms by accepting connections and then blocking on them indefinitely, causing the corresponding worm thread to block. Unfortunately, worms can avoid this mechanisms by probing and infecting asynchronously. Under the connection-throttling approach [80, 71], each host restricts the rate at which connections may be initiated. If adopted universally, such an approach would reduce the spreading rate of a worm by up to an order of magnitude, without affecting legitimate communications.

These systems are effective only against scanning worms (not topological, or "hit-list" worms), and rely on the assumption that most scans will result in non-connections. As such, they are susceptible to false positives, either accidentally (*e.g.,* when a host is joining a peer-to-peer network such as Gnutella, or during a temporary network outage) or on purpose (*e.g.,* a malicious web page with many links to images in random/not-used IP addresses). Furthermore, it may be possible for several instances of a worm to collaborate in providing the illusion of several successful connections, or to use a list of *known repliers* to blind the anomaly detector.

Another algorithm for finding fast-spreading worms using 2-level filtering based on sampling from the set of distinct source-destination pairs is described by Venkataraman *et al.* [72]. Whyte *et al.* [79] propose correlating DNS queries and replies with outgoing connections from an enterprise network to detect anomalous behavior. The main intuition is that connections due to random-scanning (and, to a degree, hit-list) worms will not be preceded by DNS transactions. This approach can be used to detect other types of malicious behavior, such as mass-mailing worms and network reconnaissance.

Kim and Karp [32] describe an algorithm for correlating packet payloads from different traffic flows, toward deriving a worm signature, by using a simple portscan-based traffic flow classifier to limit the amount of traffic that needs to be considered. Earlybird [63] presents a more practical algorithm for doing payload sifting, and correlates these with a range of unique sources generating infections and destinations being targeted. Another similar approach, aimed primarily at DoS traffic, is described by Matrawy *et al.* [45]. However, polymorphic and metamorphic worms [69] remain a challenge; Spinelis [66] shows that it is an NP-hard problem. Buttercup [53] attempts to detect polymorphic buffer overflow attacks by identifying the ranges of the possible return memory addresses for existing buffer overflow vulnerabilities. Unfortunately, this heuristic cannot be employed against some of the more sophisticated overflow attack tech-

niques [54]. Furthermore, the false positive rate is very high, ranging from 0.01% to 1.13%. Vigna *et al.* [73] discuss a method for testing detection signatures against mutations of known vulnerabilities to determine the quality of the detection model and mechanism. Polygraph [49] attempts to detect polymorphic exploits by identifying common invariants among the various attack instances, such as return addresses, protocol framing and poor obfuscation. Wang *et al.* [76] propose applying such content-sifting techniques within clusters of traffic, as created with header-based multi-dimensional flow clustering. In many cases, this will improve the purity of signature pools. Wang and Stolfo [78] use byte distributions (1-gram frequencies) in payloads to identify traffic that deviates from normal (based on training). While very efficient, this technique is vulnerable to worms that know, or can determine (*e.g.*, through passive monitoring), the expected normal distribution for a site and morph their attack ("blend") payload accordingly [27]. One approach to countering such mimicry efforts is to use probabilistic (randomized) payload modeling, preventing the adversary from knowing which traffic characteristics are being modeled [77]. Unfortunately, learning signatures without corroboration in an adversarial environment (*i.e.*, when the attacker can provide inputs of his choice to the learning engine) is a difficult problem, fraught with several risks [17, 50].

Polychronakis *et al.* [55] use emulation to accurately detect (potentially zero-day) polymorphic decryptors, albeit at relatively low network speeds. Bruschi *et al.* [10] use control-flow graph matching for detecting metamorphic viruses. Other work attempts to automatically reconstruct a worm's control flow from the captured attack payload [33, 16]. Crandall *et al.* [22] discuss the problem of generating quality vulnerability-specific signatures via an empirical study of the behavior of poly- and meta-morphic malcode. They outline the difficulty of identifying enough features of an exploit to generalize about a specific vulnerability. Focusing on the behavior of malcode seems to be a more promising approach. Some work has been done to generate anomaly-based signatures for web servers [60].

HoneyStat [23] runs sacrificial services inside a virtual machine, and monitors memory, disk, and network events to detect abnormal behavior. For some classes of attacks (*e.g.*, buffer overflows), this can produce highly accurate alerts with relatively few false positives, and can detect zero-day worms. Although the system only protects against scanning worms, "active honeypot" techniques [85] may be used to make it more difficult for an automated attacker to differentiate between HoneyStats and real servers. FLIPS (Feedback Learning IPS) [41] is a similar hybrid approach that incorporates a supervision framework in the presence of suspicious traffic. Instruction-set randomization is used to isolate attack vectors, which are used to train the anomaly detector. Liang and Sekar [38] and Xu *et al.* [83] concurrently propose using address space randomization to drive the detection of memory corruption vulnerabilities and create a signature to block further exploits of this type. One danger with such systems is that an attacker may be able to "train" them to consider an otherwise innocent byte stream as an attack, by including it in a series of real, mutating attacks [17]. This allows the attacker to convert a vulnerability into a self-inflicted DoS attack, if the system does not carefully analyze the suspicious inputs.

Shadow honeypots [2] combine the best features found in anomaly detectors and honeypots to create an application aware network intrusion detection system. The anomaly detectors differentiate between trusted and untrusted traffic; trusted traffic is processed normally whilst untrusted traffic is forwarded to a protected ("shadow") instance of the application. The system provides an elegant way to deal with false positives, since all requests are processed albeit some incur additional latency.

The key idea of the Rx system [57] is to checkpoint the execution of a process in anticipation of system errors. When an error is encountered, execution is rolled back and replayed, but with the process's environment changed in a way that does not violate the API's its code expects. This procedure is repeated with different environment alterations until execution proceeds past the detected error point. This procedure is a clever way to avoid the semantically incorrect fixes of failure oblivious computing and error virtualization.

The authors of [24] propose to enhance NIDS alerts using host-based IDS information. Nemean [86] is an architecture for generating semantics-aware signatures, which are signatures aware of protocol semantics (as opposed to generic byte strings).

Rajab *et al.* [58] show that a distributed worm monitor can detect non-uniform scanning worms two to four times as fast as a centralized telescope, and that knowledge of the vulnerability density of the population can further improve detection time.

Reference [48] describes a design space of worm containment systems using three parameters: reaction time, containment strategy, and deployment scenario. The authors use a combination of analytic modeling and simulation to describe how each of these design factors impacts the dynamics of a worm epidemic. Their analysis suggests that there are significant gaps in containment defense mechanisms that can be employed, and that considerably more research (and better coordination between ISPs and other entities) is needed. Their analysis focuses exclusively on containment mechanisms (*i.e.*, network filtering), which they consider the only viable defense mechanism. We believe that other types of automated defense mechanisms will eventually be invented, if only because containment mechanisms can severely impact service availability.

Shield [75] is a mechanism for pushing to workstations vulnerability-specific, application-aware filters expressed as programs in a simple language. These programs roughly mirror the state of the protected service, allowing for more intelligent application of content filters, as opposed to simplistic payload string matching.

Wang *et al.* [74] presented encouraging results for slowing down the spread of viruses. The authors simulated the propagation of virus infections through certain types of net-

works, coupled with partial immunization. Their findings show that even with low levels of immunization, the infection slows down significantly. Those experiments looked at a single virus. Our work investigates the detection of multiple viruses when there is no *a priori* knowledge of which viruses may attack.

One approach for detecting new email viruses was described by Bhattacharyya *et al.* [7], which keeps track of email attachments as they are exchanged between users via a set of collaborating email servers that forward a subset of their data to a central data warehouse and correlation server. Only attachments with a high frequency of appearance are deemed suspicious; furthermore, the email exchange patterns among users are used to create models of normal behavior. Deviation from such behavior (*e.g.,* a user sending a particular attachment to a large number of other users at the same site, to which she has never sent email before) raises an alarm. Information about dangerous attachments can be sent to the email servers, which then filter these out. One interesting result is that their system only need be deployed to a small number of email servers, such that it can examine a miniscule amount of email traffic (relative to all email exchanged on the Internet) — they claim 0.1% — before they can determine virus outbreaks and be able to build good user behavior models. A similar technique, tracking attachments through the network, is described by Xiong [82]. An attempt to apply behavior-based detection at the network layer for worm detection is discussed by Ellis *et al.* [26]. A high-level architecture for building behavior profiles for hosts in the absence of a persistent identity is described by Allman *et al.* [1].

Locasto *et al.* [40] propose coordinating the sharing of IDS alerts for detecting worm attacks and portscanning across administrative domains. A similar approach is presented by Li *et al.* [37]. Parekh *et al.* [52] describe several methods for privacy-preserving correlation of payloads across multiple sensors to identify attacks. Malan and Smith correlate traces of system calls (in terms of frequency distributions) across different machines to detect abnormal behavior [43]. Cheetancheri *et al.* [14] experimentally investigate the parameters of a cooperating alert sharing protocol coupled with distributed sequential hypothesis testing to generate global alarms about distributed attacks. Briesenmeister and Porras [9] use formal methods and a model of collaborative worm defenses to create propagation strategies that prevent such defenses from reaching global consensus. They conclude that randomized algorithms offer one approach toward foiling this strategy, citing COVERAGE as a specific example.

Vigilante [20, 21] proposes the concept of Self-Certifying Alerts, which are exchanged between hosts as a result of a newly detected attack. The recipient can verify the validity of the alert and use an appropriate protection mechanism. Vigilante supplies a mechanism to detect an exploited vulnerability (by analyzing the control flow path taken by executing injected code) and defines a data structure (Self-Certifying Alert) for exchanging information about this discovery. A major advantage of this vulnerability-specific approach is that Vigilante is exploit-agnostic and can be used to defend against polymorphic worms. Porras *et al.* [56] argue that hybrid defenses using complementary techniques (in their case, connection throttling at the domain gateway and a peer-based coordination mechanism), can be much more effective against worms.

Reference [70] proposes the use of "predator" viruses that spread in much the same way malicious viruses do but try to eliminate their designated "victim" viruses. The authors show that predators can be made to perform their tasks without flooding the network and consuming all available resources. However, designers of predators would have to find their own exploits (or safeguard exploits for future use), which is not an attractive proposition. Furthermore, many recent worms have been closing the hole they exploited, after infecting a machine.

DOMINO [84] is an overlay system for cooperative intrusion detection. The system is organized in two layers, with a small core of trusted nodes and a larger collection of nodes connected to the core. The experimental analysis demonstrates that a coordinated approach has the potential of providing early warning for large-scale attacks while reducing potential false alarms. The system described in [11] similarly uses a DHT-based overlay network to automatically correlate all relevant information.

There are two key differences between DOMINO-like systems and COVERAGE. First, COVERAGE provides much broader coverage in terms of collecting data from different sites, while DOMINO is restricted to monitoring unused address-space (which could be avoided by a smart attacker). Second, COVERAGE considers all participants to be untrusted, avoiding the introduction of a trusted core overlay as this kind of infrastructure could be easily attacked by a worm (*i.e.,* in the early stages of the infection).

Cooke *et al.* [19] describe an architecture for identifying already compromised hosts that are part of a botnet. Their approach uses monitoring of common communication channels used to disseminate control directives to bots (such as IRC), as well as other traffic-based detection heuristics. Another IRC-based detection algorithm is described by Binkley and Singh [8]. Ramachandran *et al.* [59] monitor the DNS-blackhole reconnaissance, used by the bot controllers to identify whether specific bots can send spam, for identifying likely bots.

Reference [88] describes an architecture and models for an early warning system, where the participating nodes and routers propagate alarm reports towards a centralized site for analysis. The question of how to respond to alerts is not addressed, and, similar to DOMINO, the use of a centralized collection and analysis facility is weak against worms attacking the early warning infrastructure.

The earliest work on cooperative response mechanisms is that of Nojiri *et al.* [51]. They present a cooperative response algorithm where edge-routers share attack reports with a small set of other edge-routers. Edge-routers update their alert level based on the shared attack reports and decide whether to enable traffic filtering and blocking for a

particular attack. Analysis by Kannan *et al.* [30] has shown that cooperative response algorithms can improve containment, even when a minority of firewalls cooperate(for example, even when fewer than 10% of the firewalls cooperate, a cooperative scheme can provide 95% containment — even in the face of hundreds of malicious firewalls). That work, however promising, does not directly relate to our work. They are more concerned with a single fast virus — the analysis focuses on a single virus (consequently underplaying the cost of over-aggressive response), has a weaker model of "malicious" firewalls (malicious firewalls merely stay silent, but do not mislead through false alarms), and does not explore the benefits of allowing more lattitude in generating false alarms.

6 Conclusions and Future Plans

We have described an algorithm, named COVERAGE, that allows cooperating agents to share information about the spread of malicious virus in the Internet and use this information for controlling the behavior of detection and filtering resources. The algorithm operates without fully trusting such information, so as to limit the damage of false alarms injected by malicious or faulty nodes. Our solution is based on the idea of carefully sampling of global state to validate claims made by individual participants. Simulation results confirm that this method is effective in limiting the damage of virus attacks, and that it is robust against attacks by malicious participants.

Our approach is motivated by the belief that for day-zero virus detectors to be maximally effective, they need the latitude to be paranoid and flag even mildly suspicious behavior as a possible attack. Consequently, we focus on how how well defensive systems can deal with frequent false alarms, and whether they can sensibly allocate resources to cover many simultaneous attacks. In this respect, COVERAGE performs well. When compared against a similar approach, the NRL03 algorithm [51], COVERAGE exhibits a lower cost in terms of scanning for worms due to its resource-aware approach. Furthermore, it has a lower communication cost in the presence of false alarms and fast worms, and can detect and react to slow-propagating worms better. However, the price COVERAGE pays for screening for false alarms is that it may not react as quickly to fast worms. The price it pays for reacting to slow worms more quickly is higher communication overhead than NRL03 for slow worms.

Acknowledgements This work was supported in part by the National Science Foundation under grants CT CNS-06-27473, ITR CNS-04-26623, DUE-04-17085, and CCR-03-31584.

References

1. Allman, M., Blanton, E., Paxson, V.: An Architecture for Developing Behavioral History. In: Proceedings of the 8^{th} Information Security Conference (ISC) (2005)
2. Anagnostakis, K., Sidiroglou, S., Akritidis, P., Xinidis, K., Markatos, E., Keromytis, A.D.: Detecting Targetted Attacks Using Shadow Honeypots. In: Proceedings of the 14^{th} USENIX Security Symposium, pp. 129–144 (2005)
3. Anagnostakis, K.G., Greenwald, M.B., Ioannidis, S., Keromytis, A.D., Li, D.: A Cooperative Immunization System for an Untrusting Internet. In: Proceedings of the 11^{th} IEEE Internation Conference on Networking (ICON), pp. 403–408 (2003)
4. Anagnostakis, K.G., Greenwald, M.B., Ioannidis, S., Miltchev, S.: Open Packet Monitoring on FLAME: Safety, Performance and Applications. In: Proceedings of the 4^{th} International Working Conference on Active Networks (2002)
5. Antonatos, S., Akritidis, P., Markatos, E.P., Anagnostakis, K.G.: Defending against Hitlist Worms using Network Address Space Randomization. In: Proceedings of the ACM Workshop on Rapid Malcode (WORM), pp. 30–40 (2005)
6. Bailey, M., Cooke, E., Jahanian, F., Watson, D., Nazario, J.: The Blaster Worm: Then and Now. IEEE Security & Privacy **3**(4), 26–31 (2005)
7. Bhattacharyya, M., Schultz, M.G., Eskin, E., Hershkop, S., Stolfo, S.J.: MET: An Experimental System for Malicious Email Tracking. In: Proceedings of the New Security Paradigms Workshop (NSPW), pp. 1–12 (2002)
8. Binkley, J.R., Singh, S.: An Algorithm for Anomaly-based Botnet Detection. In: Proceedings of Steps to Reducing Unwanted Traffic on the Internet Workshop (SRUTI), pp. 43–48 (2006)
9. Briesenmeister, L., Porras, P.A.: Automatically Deducing Propagation Sequences that Circumvent a Collaborative Worm Defense. In: Proceedings of the 25^{th} International Performance Computing and Communications Conference (Workshop on Malware), pp. 587–592 (2006)
10. Bruschi, D., Martignoni, L., Monga, M.: Detecting Self-mutating Malware Using Control-Flow Graph Matching. In: Proceedings of the 3^{rd} International Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA), pp. 129–143 (2006)
11. Cai, M., Hwang, K., Kwok, Y.K., Song, S., Chen, Y.: Collaborative Internet Worm Containment. IEEE Security & Privacy Magazine **3**(3), 25–33 (2005)
12. CERT Advisory CA-2001-19: 'Code Red' Worm Exploiting Buffer Overflow in IIS Indexing Service DLL. http://www.cert.org/advisories/CA-2001-19.html (2001)
13. Cert Advisory CA-2003-04: MS-SQL Server Worm. `http://www.cert.org/advisories/CA-2003-04.html` (2003)
14. Cheetancheri, S.G., Agosta, J.M., Dash, D.H., Levitt, K.N., Rowe, J., Schooler, E.M.: A Distributed Host-based Worm Detection System. In: Proceedings of the SIGCOMM Workshop on Large-Scale Attack Defense (LSAD) (2006)
15. Chen, Z., Ji, C.: A Self-Learning Worm Using Importance Scanning. In: Proceedings of the ACM Workshop on Rapid Malcode (WORM), pp. 22–30 (2005)
16. Chinchani, R., Berg, E.V.D.: A Fast Static Analysis Approach to Detect Exploit Code Inside Network Flows. In: Proceedings of the 8^{th} International Symposium on Recent Advances in Intrusion Detection (RAID), pp. 284–304 (2005)
17. Chung, S.P., Mok, A.K.: Allerge Attack Against Automatic Signature Generation. In: Proceedings of the 9^{th} International Symposium on Recent Advances in Intrusion Detection (RAID), pp. 61–80 (2006)
18. Cohen, F.: Computer Viruses: Theory and Practice. Computers & Security **6**, 22–35 (1987)
19. Cooke, E., Jahanian, F., McPherson, D.: The Zombie Roundup: Understanding, Detecting, and Disrupting Botnets. In: Proceedings of the 8^{th} Information Security Conference (ISC) (2005)
20. Costa, M., Crowcroft, J., Castro, M., Rowstron, A.: Can We Contain Internet Worms? In: Proceedings of the 3^{rd} Workshop on Hot Topics in Networks (HotNets) (2004)

21. Costa, M., Crowcroft, J., Castro, M., Rowstron, A.: Vigilante: End-to-End Containment of Internet Worms. In: Proceedings of the Symposium on Systems and Operating Systems Principles (SOSP) (2005)
22. Crandall, J.R., Su, Z., Wu, S.F., Chong, F.T.: On Deriving Unknown Vulnerabilities from Zero-Day Polymorphic and Metamorphic Worm Exploits. In: Proceedings of the 12^{th} ACM Conference on Computer and Communications Security (CCS), pp. 235–248 (2005)
23. Dagon, D., Qin, X., Gu, G., Lee, W., Grizzard, J., Levine, J., Owen, H.: HoneyStat: Local Worm Detection Using Honepots. In: Proceedings of the 7^{th} International Symposium on Recent Advances in Intrusion Detection (RAID), pp. 39–58 (2004)
24. Dreger, H., Kreibich, C., Paxson, V., Sommer, R.: Enhancing the Accuracy of Network-based Intrusion Detection with Host-based Context. In: Proceedings of the Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA) (2005)
25. Dubendorfer, T., Wagner, A., Hossmann, T., Plattner, B.: Flow-Level Traffic Analysis of the Blaster and Sobig Worm Outbreaks in an Internet Backbone. In: Proceedings of the Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA) (2005)
26. Ellis, D.R., Aiken, J.G., Attwood, K.S., Tenaglia, S.D.: A Behavioral Approach to Worm Detection. In: Proceedings of the ACM Workshop on Rapid Malcode (WORM), pp. 43–53 (2004)
27. Fogla, P., Sharif, M., Perdisci, R., Kolesnikov, O., Lee, W.: Polymorphic Blending Attacks. In: Proceedings of the 15^{th} USENIX Security Symposium, pp. 241–256 (2006)
28. Goel, S., Bush, S.F.: Biological Models of Security for Virus Propagation in Computer Networks. USENIX *;login:* **29**(6), 49–56 (2004)
29. Jung, J., Paxson, V., Berger, A.W., Balakrishnan, H.: Fast Portscan Detection Using Sequential Hypothesis Testing. In: Proceedins of the IEEE Symposium on Security and Privacy (2004)
30. Kannan, J., Subramanian, L., Stoica, I., Katz, R.H.: Analyzing Cooperative Containment of Fast Scanning Worms. In: Proceedings of Steps to Reducing Unwanted Traffic on the Internet Workshop (SRUTI), pp. 17–23 (2005)
31. Kephart, J.O.: A Biologically Inspired Immune System for Computers. In: Artificial Life IV: Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems, pp. 130–139. MIT Press (1994)
32. Kim, H., Karp, B.: Autograph: Toward Automated, Distributed Worm Signature Detection. In: Proceedings of the 13^{th} USENIX Security Symposium, pp. 271–286 (2004)
33. Krugel, C., Kirda, E., Mutz, D., Robertson, W., Vigna, G.: Polymorphic Worm Detection Using Structural Information of Executables. In: Proceedings of the 8^{th} International Symposium on Recent Advances in Intrusion Detection (RAID), pp. 207–226 (2005)
34. Leavitt, N.: Mobile Phones: The Next Frontier for Hackers? IEEE Computer **38**(4) (2005)
35. Levine, J.G., Grizzard, J.B., Owen, H.L.: Using Honeynets to Protect Large Enterprise Networks. IEEE Security & Privacy **2**(6), 73–75 (2004)
36. Levy, E.: Approaching Zero. IEEE Security & Privacy **2**(4), 65–66 (2004)
37. Li, Z., Chen, Y., Beach, A.: Towards Scalable and Robust Distributed Intrusion Alert Fusion with Good Load Balancing. In: Proceedings of the 1^{st} Workshop on Large-Scale Attack Defence (LSAD), pp. 115–122 (2006)
38. Liang, Z., Sekar, R.: Fast and Automated Generation of Attack Signatures: A Basis for Building Self-Protecting Servers. In: Proceedings of the 12^{th} ACM Conference on Computer and Communications Security (CCS), pp. 213–222 (2005)
39. Liston, T.: Welcome To My Tarpit: The Tactical and Strategic Use of LaBrea. `http://www.threenorth.com/LaBrea/LaBrea.txt` (2001)
40. Locasto, M., Parekh, J., Stolfo, S., Keromytis, A., Malkin, T., Misra, V.: Collaborative Distributed Intrusion Detection. Tech. Rep. CUCS-012-04, Columbia University Department of Computer Science (2004)
41. Locasto, M., Wang, K., Keromytis, A., Stolfo, S.: FLIPS: Hybrid Adaptive Intrusion Prevention. In: Proceedings of the 8^{th} Symposium on Recent Advances in Intrusion Detection (RAID) (2005)
42. Ma, J., Voelker, G.., Savage, S.: Self-Stopping Worms. In: Proceedings of the ACM Workshop on Rapid Malcode (WORM), pp. 12–21 (2005)
43. Malan, D.J., Smith, M.D.: Host-Based Detection of Worms through Peer-to-Peer Cooperation. In: Proceedings of the ACM Workshop on Rapid Malcode (WORM), pp. 72–80 (2005)
44. Mannan, M., van Oorschot, P.C.: On Instant Messaging Worms, Analysis and Countermeasures. In: Proceedings of the ACM Workshop on Rapid Malcode (WORM), pp. 2–11 (2005)
45. Matrawy, A., van Oorschot, P.C., Somayaji, A.: Mitigating Network Denial-of-Service Through Diversity-Based Traffic Management. In: Proceedings of the 3^{rd} International Conference on Applied Cryptography and Network Security (ACNS), pp. 104–121 (2005)
46. Miretskiy, Y., Das, A., Wright, C.P., Zadok, E.: Avfs: An On-Access Anti-Virus File System. In: Proceedings of the 13^{th} USENIX Security Symposium, pp. 73–88 (2004)
47. Moore, D., Shanning, C., Claffy, K.: Code-Red: a case study on the spread and victims of an Internet worm. In: Proceedings of the 2nd Internet Measurement Workshop, pp. 273–284 (2002)
48. Moore, D., Shannon, C., Voelker, G., Savage, S.: Internet Quarantine: Requirements for Containing Self-Propagating Code. In: Proceedings of 22^{nd} Annual Joint Conference of IEEE Computer and Communication societies (INFOCOM) (2003)
49. Newsome, J., Karp, B., Song, D.: Polygraph: Automatically Generating Signatures for Polymorphic Worms. In: Proceedings of the IEEE Security & Privacy Symposium, pp. 226–241 (2005)
50. Newsome, J., Karp, B., Song, D.: Paragraph: Thwarting Signature Learning by Training Maliciously. In: Proceedings of the 9^{th} International Symposium on Recent Advances in Intrusion Detection (RAID), pp. 81–105 (2006)
51. Nojiri, D., Rowe, J., Levitt, K.: Cooperative response strategies for large scale attack mitigation. In: Proceedings of the 3^{rd} DARPA Information Survivability Conference and Exposition (2003)
52. Parekh, J.J., Wang, K., Stolfo, S.J.: Privacy-Preserving Payload-Based Correlation for Accurate Malicious Traffic Detection. In: Proceedings of the 1^{st} Workshop on Large-Scale Attack Defence (LSAD), pp. 99–106 (2006)
53. Pasupulati, A., Coit, J., Levitt, K., Wu, S., Li, S., Kuo, J., Fan, K.: Buttercup: On Network-based Detection of Polymorphic Buffer Overflow Vulnerabilities. In: Proceedings of the Network Operations and Management Symposium (NOMS), pp. 235–248, vol. 1 (2004)
54. Pincus, J., Baker, B.: Beyond Stack Smashing: Recent Advances in Exploiting Buffer Overflows. IEEE Security & Privacy **2**(4), 20–27 (2004)
55. Polychronakis, M., Anagnostakis, K.G., Markatos, E.: Network-Level Polymorphic Shellcode Detection Using Emulation. In: Proceedings of the 3^{rd} International Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA), pp. 54–73 (2006)
56. Porras, P., Briesemeister, L., Levitt, K., Rowe, J., Ting, Y.C.A.: A Hybrid Quarantine Defense. In: Proceedings of the ACM Workshop on Rapid Malcode (WORM), pp. 73–82 (2004)
57. Qin, F., Tucek, J., Sundaresan, J., Zhou, Y.: Rx: Treating Bugs as Allergies – A Safe Method to Survive Software Failures. In: Proceedings of the Symposium on Systems and Operating Systems Principles (SOSP) (2005)
58. Rajab, M.A., Monrose, F., Terzis, A.: On the Effectiveness of Distributed Worm Monitoring. In: Proceedings of the 14^{th} USENIX Security Symposium, pp. 225–237 (2005)

59. Ramachandran, A., Feamster, N., Dagon, D.: Revealing Botnet Membership Using DNSBL Counter-Intelligence. In: Proceedings of Steps to Reducing Unwanted Traffic on the Internet Workshop (SRUTI), pp. 49–54 (2006)
60. Robertson, W., Vigna, G., Kruegel, C., Kemmerer, R.A.: Using Generalization and Characterization Techniques in the Anomaly-based Detection of Web Attacks. In: Proceedings of the 13^{th} Symposium on Network and Distributed System Security (NDSS) (2006)
61. Shannon, C., Moore, D.: The Spread of the Witty Worm. IEEE Security & Privacy **2**(4), 46–50 (2004)
62. Sidiroglou, S., Keromytis, A.D.: A Network Worm Vaccine Architecture. In: Proceedings of the IEEE Workshop on Enterprise Technologies: Infrastructure for Collaborative Enterprises (WET-ICE), Workshop on Enterprise Security, pp. 220–225 (2003)
63. Singh, S., Estan, C., Varghese, G., Savage, S.: Automated worm fingerprinting. In: Proceedings of the 6^{th} Symposium on Operating Systems Design & Implementation (OSDI) (2004)
64. The Spread of the Sapphire/Slammer Worm. http://www.silicondefense.com/research/worms/slammer.php (2003)
65. Song, D., Malan, R., Stone, R.: A Snapshot of Global Internet Worm Activity. Tech. rep., Arbor Networks (2001)
66. Spinellis, D.: Reliable identification of bounded-length viruses is NP-complete. IEEE Transactions on Information Theory **49**(1), 280–284 (2003). DOI doi:10.1109/TIT.2002.806137. URL `http://www.dmst.aueb.gr/dds/pubs/jrnl/2002-ieeetit-npvirus/html/npvirus.html`
67. Staniford, S., Moore, D., Paxson, V., Weaver, N.: The Top Speed of Flash Worms. In: Proceedings of the ACM Workshop on Rapid Malcode (WORM), pp. 33–42 (2004)
68. Staniford, S., Paxson, V., Weaver, N.: How to Own the Internet in Your Spare Time. In: Proceedings of the USENIX Security Symposium, pp. 149–167 (2002)
69. Ször, P., Ferrie, P.: Hunting for Metamorphic. Tech. rep., Symantec Corporation (2003)
70. Toyoizumi, H., Kara, A.: Predators: Good Will Mobile Codes Combat against Computer Viruses. In: Proceedings of the New Security Paradigms Workshop (NSPW), pp. 13–21 (2002)
71. Twycross, J., Williamson, M.M.: Implementing and testing a virus throttle. In: Proceedings of the 12th USENIX Security Symposium, pp. 285–294 (2003)
72. Venkataraman, S., Song, D., Gibbons, P.B., Blum, A.: New Streaming Algorithms for Fast Detection of Superspreaders. In: Proceedings of the 12^{th} ISOC Symposium on Network and Distributed Systems Security (SNDSS), pp. 149–166 (2005)
73. Vigna, G., Robertson, W., Balzarotti, D.: Testing Network-based Intrusion Detection Signatures Using Mutant Exploits. In: Proceedings of the 11^{th} ACM Conference on Computer and Communications Security (CCS), pp. 21–30 (2004)
74. Wang, C., Knight, J.C., Elder, M.C.: On Computer Viral Infection and the Effect of Immunization. In: Proceedings of the 16th Annual Computer Security Applications Conference, pp. 246–256 (2000)
75. Wang, H.J., Guo, C., Simon, D.R., Zugenmaier, A.: Shield: Vulnerability-Driven Network Filters for Preventing Known Vulnerability Exploits. In: Proceedings of the ACM SIGCOMM Conference, pp. 193–204 (2004)
76. Wang, J., Hamadeh, I., Kesidis, G., Miller, D.J.: Polymorphic Worm Detection and Defense: System Design, Experimental Methodology, and Data Resources. In: Proceedings of the 1^{st} Workshop on Large-Scale Attack Defence (LSAD), pp. 169–176 (2006)
77. Wang, K., Parekh, J., Stolfo, S.J.: ANAGRAM: A Content-Based Anomaly Detector Resistant to Mimicry Attack. In: Proceedings of the 9^{th} International Symposium on Recent Advances in Intrusion Detection (RAID), pp. 226–248 (2006)
78. Wang, K., Stolfo, S.J.: Anomalous Payload-based Network Intrusion Detection. In: Proceedings of the 7^{th} International Symposium on Recent Advances in Intrusion Detection (RAID), pp. 201–222 (2004)
79. Whyte, D., Kranakis, E., van Oorschot, P.: DNS-based Detection of Scanning Worms in an Enterprise Network. In: Proceedings of the 12^{th} ISOC Symposium on Network and Distributed Systems Security (SNDSS), pp. 181–195 (2005)
80. Williamson, M.: Throttling Viruses: Restricting Propagation to Defeat Malicious Mobile Code. Tech. Rep. HPL-2002-172, HP Laboratories Bristol (2002)
81. Wu, J., Vangala, S., Gao, L., Kwiat, K.: An Effective Architecture and Algorithm for Detecting Worms with Various Scan Techniques. In: Proceedings of the Network and Distributed System Security (NDSS) Symposium, pp. 143–156 (2004)
82. Xiong, J.: ACT: Attachment Chain Tracing Scheme for Email Virus Detection and Control. In: Proceedings of the ACM Workshop on Rapid Malcode (WORM), pp. 11–22 (2004)
83. Xu, J., Ning, P., Kil, C., Zhai, Y., Bookholt, C.: Automatic Diagnosis and Response to Memory Corruption Vulnerabilities. In: Proceedings of the 12^{th} ACM Conference on Computer and Communications Security (CCS), pp. 222–234 (2005)
84. Yegneswaran, V., Barford, P., Jha, S.: Global Intrusion Detection in the DOMINO Overlay System. In: Proceedings of NDSS (2004)
85. Yegneswaran, V., Barford, P., Plonka, D.: On the Design and Use of Internet Sinks for Network Abuse Monitoring. In: Proceedings of the 7^{th} International Symposium on Recent Advances in Intrusion Detection (RAID), pp. 146–165 (2004)
86. Yegneswaran, V., Giffin, J.T., Barford, P., Jha, S.: An Architecture for Generating Semantics-Aware Signatures. In: Proceedings of the 14^{th} USENIX Security Symposium, pp. 97–112 (2005)
87. Zhou, L., Zhang, L., Sherry, F.M., Immorlica, N., Costa, M., Chien, S.: A First Look at Peer-to-Peer Worms: Threats and Defenses. In: Proceedings of the 4^{th} International Workshop on Peer-To-Peer Systems (IPTPT) (2005)
88. Zou, C.C., Gao, L., Gong, W., Towsley, D.: Monitoring and Early Warning for Internet Worms. In: Proceedings of the 10^{th} ACM International Conference on Computer and Communications Security (CCS), pp. 190–199 (2003)
89. Zou, C.C., Gong, W., Towsley, D.: Code Red Worm Propagation Modeling and Analysis. In: Proceedings of the 9th ACM Conference on Computer and Communications Security, pp. 138–147 (2002)

Author Biographies

Dr. Kostas Anagnostakis is a Principal Investigator on software and systems security at the Institute for Infocomm Research (I2R) in Singapore. He holds a Ph.D. degree in Computer and Information Science from the University of Pennsylvania, USA, a Master's degree from the same school and a B.Sc. in Computer Science from the University of Crete. His main areas of interest are in distributed systems security, networking, performance evaluation, and in problems that lie at the intersection between computer science and economics.

Michael Greenwald is a Research Computer Scientist at Alcatel-Lucent Bell Laboratories and an Adjunct Assistant Professor of Computer and Information Science at the University of Pennsylvania. Michael received his PhD degree in Computer Science from Stanford University, and his Bachelors (in Mathematics) from MIT. One of his research interests is in techniques that enhance cooperation, coordination, adaptability, and availability in distributed systems.

Sotiris Ioannidis received his Ph.D. in Computer Science from the University of Pennsylvania, Philadelphia, PA, in

2005. He received his M.S. in Computer Science from the University of Rochester, NY, in 1997 and his B.S. in Mathematics from the University of Crete, Heraclion, Greece, in 1996. Currently he is a Research Scholar in the Computer Science Department of Stevens Institute of Technology. His research interests are in host and network security, operating and distributed systems, and security policies in large systems.

Angelos Keromytis is an Associate Professor of Computer Science at Columbia University. He received his Masters and PhD from the University of Pennsylvania, and his Bachelors (all in Computer Science) from the University of Crete, in Greece. His research interests include network and system survivability, authorization and access control, and large-scale systems security. A full CV can be found at `http://www.cs.columbia.edu/~angelos/cv.html`